Applied Mathematical Sciences, Vol. 9, 2015, no. 62, 3097 - 3103
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ams.2015.52169

Modified Skala's Plane Tested Algorithm for Line – Polyhedron Intersection

A. M. Konashkova

Ural Federal University
19 Mira St., Ekaterinburg, 620002, Russia

Abstract

Three modifications of known Skala's line clipping algorithm are presented. Basic Skala's algorithm represents the line as two intersected planes. Line intersects a triangular face of polyhedron only if both planes intersect the triangle. Following along the first plane triangles tested against the second plane and then line-triangle or line-half space intersection is tested. First modification consists in saving some temporary variables. Second modification consists in choosing of direction from the first triangle. Third modification consists in utilization of some precomputed values and a novel line-triangle intersection test. Three modifications give up to 26 % performance gain.

Keywords: line, polyhedron, intersection, plane tested algorithm

1 Introduction

Intersection of lines, rays and segments against various geometrical objects is widely used in radiative heat transfer, computational geometry and computer graphics. In radiative heat transfer such objects are metal bars, furnace walls and mechanical assemblies. In computer graphics such objects are buildings, interior objects and animated characters [4]. Line/Ray – polyhedron intersection is central problem in ray tracing for rendering and also for radiation obstruction modeling [7]. Usually only polyhedra with triangular faces are considered.

There are two most useful algorithms: direct computational algorithm and the Cyrus-Beck algorithm. Both of them are very popular up to date.

The direct computational algorithm [2] performs direct line – triangle intersection [3] for each triangular face of the given polyhedron while two intersections are not found.

The Cyrus-Beck algorithm [1] uses the fact that a convex polyhedron can be understood as the intersection of half spaces. Boundaries of these half spaces are formed by planes in which faces of the polyhedron lie. Suppose one have a convex polyhedron and a line with some parametrization. Searching for the intersection of these geometrical objects, one can divide the bounding planes of the polyhedron into two groups according to the orientation of their normal vectors. Among the planes oriented towards the observer, one search for the point of intersection with the maximal parameter value t_1. Among planes of the other group, the minimal parameter value t_2 is found. If $t_1 > t_2$, the intersection of the polyhedron with the given line does not exist. If $t_1 \leq t_2$ intersection points are computed [1]. In other words the algorithm performs line – half space intersection for each face while there is an intersection (segment).

2 Skala's algorithm

The main idea to accelerate two early algorithms is to reject polyhedron faces before the main computing. The one of possible ways is to test line – bounding volume intersection first. However, this strategy is applied usually to all geometry but not to individual polyhedron faces.

Another idea was proposed by V. Skala in [5] for triangular faces. A line L_1 can be defined as an intersection of two nonparallel planes p_1 and p_2 [5]. If the line L_1 intersects the given triangle then planes p_1 and p_2 intersect the given triangle, too, but if planes p_1 and p_2 intersect the triangle then the line can intersect (line L_1 and planes p_1 and p_2) or miss the triangle (line L_2 and planes p_3 and p_4), see fig.1. Then one can test each triangle of the given polyhedron against p_1 and p_2 planes before detailed line – triangle or line – half space intersection computation. If both planes intersect the given triangle (facet) then use detailed intersection test. The intersection of the given plane p_i and the triangle exists if and only if two vertices $\mathbf{x_j}$ and $\mathbf{x_k}$ of the triangle exist so that $sign(F_i(x_j)) \neq sign(F_i(x_k))$, where $F_i = 0$ is an equation for the *i*-th plane p_i, *i=1,2*.

This test can be applied with direct computational algorithm or with Cyrus-Beck algorithm and computing time may be decreased 2-4 times for 100 polyhedron faces or 34 – 6.1 times for 1000 polyhedron faces [2].

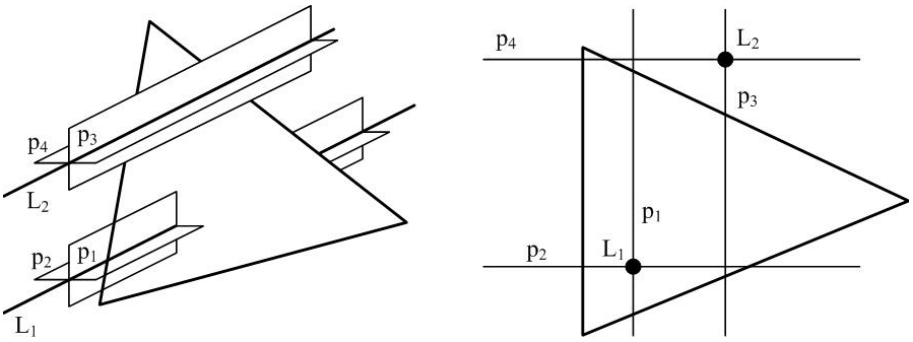

Fig. 1 Usage of two planes for line definition

The rejection test allows one to determine an edge intersected by planes p_1 or p_2 and next triangle shared by this edge. Thus one can test not all triangles against plains p_1 and p_2, but only ring of triangles which intersected by plane p_1. By following from one triangle to next triangle with common edge (see fig. 2a) we can test only $O(\sqrt{N})$ triangles. This algorithm described in details in [6].

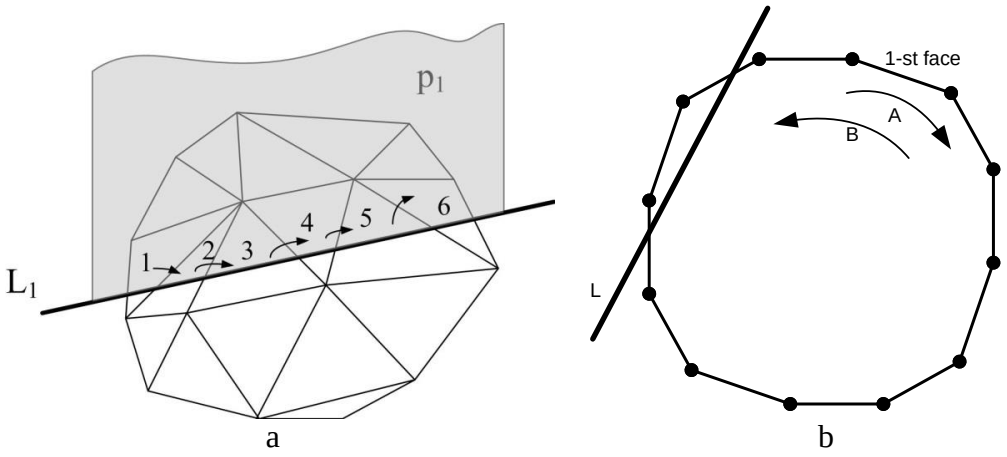

Fig. 2 Testing sequence of triangles with common edge: a) 3d view; b) choosing the direction

For each triangle its vertices and neighboring triangles are known. The algorithm:

1) Choose 1-st triangle and calculate its center C.

2) Calculate parameters of the 1-st plane $F_1 = 0$ such that line L and C lie at the plane.

3) Calculate the 2-nd plane $F_2 = 0$, $F_2 \perp F_1$. L also lies at $F_2 = 0$.

4) For triangle vertices calculate distances from the planes: $F_1(x_1), F_1(x_2), F_1(x_3), F_2(x_1), F_2(x_2), F_2(x_3)$ and their signs ($F_2(x_3)$ is calculated only if $\text{sign}F_2(x_1) = \text{sign}F_2(x_2)$, because otherwise 2-nd plane already intersects the triangle).

5) If $\mathrm{sign}F_2(x_1) = \mathrm{sign}F_2(x_2) = \mathrm{sign}F_2(x_3)$ then 2-nd plane doesn't intersect the triangle. Go to step 8.
6) Calculate line-triangle intersection [3] and count number of intersections.
7) If there are 2 intersections – line intersects the polyhedron. Reorder (if needed) points of intersection and terminate.
8) Choose next triangle: find 2 neighboring triangles situated on the opposite sides of the first one. Choose new one different from the previous tested triangle.
9) If next triangle differs from the 1-st then go to step 4, otherwise terminate (following along the ring of triangles process returns to the beginning). Update indexes of current and previous triangles.

3. Modified algorithm

3.1 Modification 1

Not all values $F_1(x_1), F_1(x_2), F_1(x_3), F_2(x_1), F_2(x_2), F_2(x_3)$, but only 2 new values $F_1(x_{nexti}), F_2(x_{nexti})$ have to be calculated because for two incident triangles 2 vertices are the same. Steps 1-8 are unchanged.

Step 9. For each vertex of next triangle $Vnext_i$ and each vertex of current triangle $Vcur_j$ test if $Vnext_i = Vcur_j$ then copy $F_1(x_{curj}), F_2(x_{curj})$ to $F_1(x_{nexti}), F_2(x_{nexti})$. If $Vnext_i \neq Vcur_j$ for j=1,2,3 then $F_1(x_{nexti}), F_2(x_{nexti})$ are recalculated.

Step 10. If next triangle differs from the 1-st then go to step 5, otherwise terminate. Update indexes of current and previous triangles.

3.2 Modification 2

Previously described algorithm of choosing the next triangle is executed all the times except the first time. As can be seen at fig. 2b second triangle can be chosen by two ways: A or B. B choice is better because less number of triangles will be tested before two intersections will be founded. So, choosing procedure for the 2-nd triangle is the following. For example, vertex 3 is on the opposite side of $F_1 = 0$ than vertex 1 and 2. Then if vertex 1 is closer to $F_2 = 0$ then next triangle with coincident vertices 3 and 1 is chosen, otherwise with 3 and 2. Corresponding algorithm is used in cases if vertex 1 or vertex 2 is distinct from others.

3.3 Modification 3

Since $F_1(x_1), F_1(x_2), F_1(x_3), F_2(x_1), F_2(x_2), F_2(x_3)$ are already known, more efficient line-triangle intersection test can be proposed than the standard Müller-Trumbore algorithm [3].

Consider the case if vertices 2 and 3 are situated on first side of $F_2 = 0$ and vertex 1 is situated on the opposite side (fig. 3). Edges $[x_1,x_2]$ and $[x_1,x_3]$ are intersected by 2-nd plane respectively at points P_{12} and P_{13}. Initial line intersects the triangle if these two points are on different sides of 1-st plane, i.e. if $\mathrm{sign}F_1(P_{12}) \neq \mathrm{sign}F_1(P_{13})$. Let us consider triangle vertices and points P_{12}, P_{13} in basis F_1, F_2 . Segments $[x_1,x_2]$ and $[x_1,x_3]$ are divided by $F_2 = 0$ as $|F_2(x_1)/F_2(x_2)|$ and $|F_2(x_1)/F_2(x_3)|$. Coordinate F_1 of P_{12} and P_{13} can be computed as:

$$F_1(P_{12}) = \frac{-F_2(x_1)F_1(x_2)+F_2(x_2)F_1(x_1)}{-F_2(x_1)+F_2(x_2)} \quad , \quad F_1(P_{13}) = \frac{-F_2(x_1)F_1(x_3)+F_2(x_3)F_1(x_1)}{-F_2(x_1)+F_2(x_3)}.$$

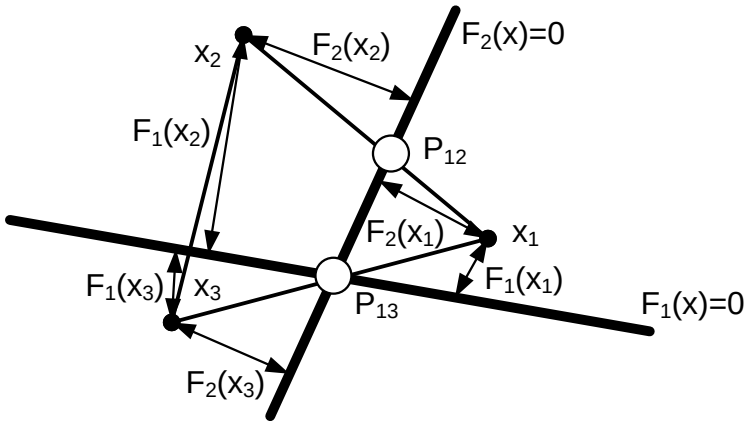

Fig. 3 Illustration of novel line-triangle intersection test

One can multiply both sub expressions by $[-F_2(x_1)+F_2(x_2)]$ and no or both signs will be changed, therefore the condition $\mathrm{sign}F_1(P_{12}) \neq \mathrm{sign}F_1(P_{13})$ will be unchanged. So, novel line-triangle intersection test is the following:

1) Calculate $S_1 = F_2(x_2)F_1(x_1) - F_2(x_1)F_1(x_2)$;
2) Calculate $S_2 = F_2(x_3)F_1(x_1) - F_2(x_1)F_1(x_3)$;
3) Test $\mathrm{sign}(S_1) \neq \mathrm{sign}(S_2)$.

If not 1-st but another vertex is on the opposite side of the 2-nd plane, the algorithm is similar to the described one. If intersection is approved line-plane intersection is executed and intersection point is calculated. Proposed algorithm requires only 4 multiplications, 2 subtractions, 2 comparisons and 1 logical operation.

4. Performance comparison

Algorithms were tested for number of polyhedron vertices *Nv* from 4 to 150. For each *Nv*, 100 randomly generated convex polyhedra were used, polyhedra

were inscribed into the bounding cube with minimal bounds x,y,z = -1 and maximal bounds x,y,z = +1. For each Nv and for each polyhedron, $2 \cdot 10^5$ lines were used, so, $2 \cdot 10^7$ lines were used for each Nv. Data sets of two points that define a line were generated such that line securely intersects the cube. Such lines and polyhedra vertices arrangement is the most right and practical one because in practice line – object intersection calculation should be always the second step after intersection the line with polyhedron bounding box. All tests were implemented in Fortran on Intel Pentium II 1.83 GHz. Relative algorithms performance is shown at fig. 4.

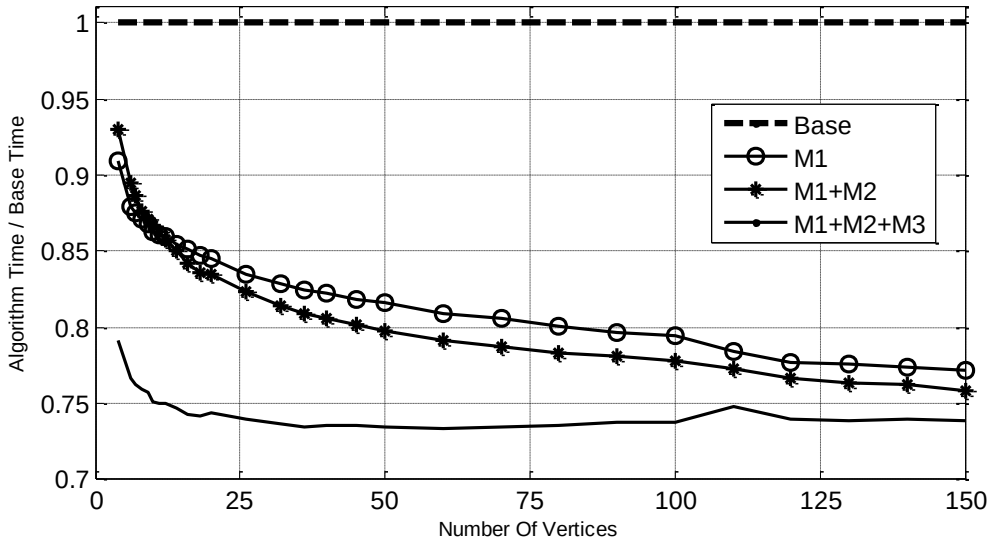

Fig. 4 Relative algorithms performance: Base – Skala's algorithm; M1 – Modification 1; M2 – Modification 2; M3 – Modification 3

It can be seen that three modifications give up to 26 % performance gain. For Nv = 4 gains are 9.1% for M1, 7.0% for M1+M2 and 20.9% for M1+M2+M3.
For Nv = 50 gains are respectively 18.4%, 20.2% and 26.5%.
For Nv = 100 gains are respectively 20.5%, 22.2% and 26.2%.
For Nv = 150 gains are respectively 22.8%, 24.2% and 26.1%.

5. Conclusions

Three modifications of known Skala's line clipping algorithm are presented and tested. Three modifications give up to 26 % performance gain. First and second modifications are applicable in both cases if line-triangle intersection or if line-half space intersection is used in Skala's algorithm. Third modification is applicable only in the first case. Possible subject for future work is testing performance of novel line-triangle tests beyond the context of Skala's algorithm.

Acknowledgements. This work was supported by Russian Foundation for Basic Research – RFBR (project № 14-08-31080).

References

[1] M. Cyrus, J. Beck, Generalized two and three dimensional clipping. Computers & Graphics, (1978), 3, 23-28.
http://dx.doi.org/10.1016/0097-8493(78)90021-3

[2] I. Kolingerova 3D - Line Clipping Algorithms - A Comparative Study, The Visual Computer, Vol.11, No.2, (1994), pp.96-104.
http://dx.doi.org/10.1007/bf01889980

[3] T. Müller and B. Trumbore, Fast, Minimum Storage Ray-Triangle Intersection, Journal of Graphics Tools, pp. 37-46, (1997), pp.21-28.
http://dx.doi.org/10.1080/10867651.1997.10487468

[4] P. Shirley, Fundamentals of computer graphics , Second Edition, AK Peters, (2005) 623 p.

[5] V. Skala, An Efficient Algorithm for Line Clipping by Convex and Non-Convex Polyhedrons in E3, Computer Graphics Forum, Vol.15, No.1, (1996), pp.61-68. http://dx.doi.org/10.1111/1467-8659.1510061

[6] V. Skala, A Fast Algorithm for Line Clipping by Convex Polyhedron in E3, Computers & Graphics, Pergamon Press, Vol.21, No.2, (1997), pp.209-214.
http://dx.doi.org/10.1016/s0097-8493(96)00084-2

[7] I. Wald, W.R. Mark, J. Gunther, S. Boulos, T. Ize et al. State of the Art in Ray Tracing Animated Scenes, Computer graphics forum, 28(6), (2009), pp. 1691-1722. http://dx.doi.org/10.1111/j.1467-8659.2008.01313.x

Received: February 5, 2015; Published: April 15, 2015